**EXPOSING THE TRUTH WITH ADVANCED FAKE NEWS DETECTION POWERED BY NATURAL LANGUAGE PROCESSING**

**Student Name:** [V.Sanjay kumar]

**Register Number:** [412323104048]

**Institution:** [Sri Ramanujar Engineeringcollege]

**Department:** [Computer Science Engineering]

**Date of Submission:** [17/05/2025]

**Github Repository Link:** [Update the project source code to your Github Repository]



# 1. Problem Statement

The explosion of digital media and online platforms has led to a significant rise in the circulation of fake news, which misleads the public, influences opinions, and causes widespread misinformation. Traditional methods of detecting such content are manual, time-consuming, and inefficient in handling the vast and rapidly growing amount of online information. There is an urgent need for an automated, accurate, and scalable solution to detect fake news.

This project aims to develop an NLP-based system that can intelligently analyze news content, identify patterns, and classify information as real or fake. By leveraging Natural Language Processing techniques, the system will enhance the credibility, reliability, and trustworthiness of information shared in the digital space.

Type of Problem:

Category: Natural Language Processing / Machine Learning

Type: Text Classification / Binary Classification Problem

Domain: Artificial Intelligence, Cybersecurity, Media & Communication

Nature: Supervised Learning (if using labeled datasets)

Keywords:

Fake News Detection

Natural Language Processing (NLP)

Text Classification

Misinformation

Supervised Learning

Machine Learning

Deep Learning

Semantic Analysis

Automated Fact-Checking

News Authenticity

Content Credibility

Social Media Monitoring

# 2. Abstract

Fake news spreads rapidly across digital platforms, misleading the public and undermining trust in authentic information sources.Traditional fact-checking methods are manual, time-consuming, and not scalable for real-time or large-scale use.

To develop an automated fake news detection system using advanced Natural Language Processing (NLP) techniques.To analyze news content linguistically and semantically for identifying and classifying it as real or fake.

Apply NLP techniques like semantic analysis, context understanding, and machine learning models (e.g., TF-IDF, BERT, LSTM).Train the system using labeled datasets of real and fake news articles to ensure high accuracy.

A scalable, real-time solution capable of detecting fake news with high precision.Reduced spread of misinformation and improved content credibility on digital platforms.Empowerment of users, journalists, and media platforms with a reliable verification tool.

Objective:

* To develop an automated fake news detection system using advanced Natural Language Processing (NLP) techniques.
* To analyze news content linguistically and semantically for identifying and classifying it as real or fake.

Approach:

Apply NLP techniques like semantic analysis, context understanding, and machine learning models (e.g., TF-IDF, BERT, LSTM).

Train the system using labeled datasets of real and fake news articles to ensure high accuracy.

Outcomes:

* A scalable, real-time solution capable of detecting fake news with high precision.
* Reduced spread of misinformation and improved content credibility on digital platforms.
* Empowerment of users, journalists, and media platforms with a reliable verification tool.

# 3. System Requirements

**1. Hardware Requirements**

**Development Environment:**

Processor: Intel i5/i7 or AMD Ryzen 5/7 (minimum), Intel i9 or AMD Ryzen 9 (recommended)

RAM: 16 GB (minimum), 32 GB or more (recommended for large datasets or models)

Storage: 512 GB SSD (minimum), 1 TB SSD recommended

GPU (Optional for Deep Learning): NVIDIA RTX 3060 or higher (CUDA support needed for frameworks like TensorFlow or PyTorch)

**Deployment Server (if self-hosted):**

CPU: Multi-core (Xeon, EPYC, or equivalent)

RAM: 32 GB or more

Storage: NVMe SSD (1 TB or more)

GPU: NVIDIA Tesla or RTX-series GPU (for real-time inference)

# Network: High-speed internet connection for accessing and verifying data sources

**2. Software Requirements**

**Operating System:**

Windows 10/11 (for development)

Ubuntu 20.04+ or Debian-based Linux (recommended for deployment)

**Development Tools:**

Python (3.8 or higher)

Jupyter Notebook or VS Code

Git and GitHub (for version control)

**Libraries & Frameworks:**

Natural Language Processing:

NLTK, SpaCy

Transformers (Hugging Face)

Gensim (for topic modeling)

**Machine Learning:**

Scikit-learn

TensorFlow or PyTorch

**Data Handling:**

Pandas, NumPy

**Model Deployment:**

Flask or FastAPI (for APIs)

Docker (for containerization)

**Web Scraping & Fact Verification:**

BeautifulSoup, Scrapy

Newspaper3k (news article extraction)

Google Fact Check Tools APIs (if needed)

**Database:**

MongoDB or PostgreSQL (for storing articles, predictions, metadata)

**Optional Tools:**

ElasticSearch (for indexing and fast text search)

RabbitMQ/Kafka (for processing pipelines in real-time detection systems)

Hugging Face Model Hub (for pretrained fake news detection models)

# 4. Objectives

Identify and classify news as fake or real using advanced NLP techniques. Extract named entities, sentiments, and topics from news articles for deeper understanding. Monitor the spread and origin of fake news across digital platforms. Provide explainable AI predictions to enhance trust and transparency. Support journalists and fact-checkers with actionable, data-driven insights. Continuously improve detection models using new data and feedback.

Binary classification label: Fake or Real.

Prediction confidence score (e.g., 93% likely fake).

Named Entity Recognition (NER) output (e.g., persons, places, organizations).

Topic classification (e.g., politics, health, finance).

Sentiment analysis result (e.g., negative, neutral, positive).

Explanation of model decision (e.g., flagged phrases or patterns).

Visualization of story propagation across sources (if implemented).

Common traits and patterns in fake news (e.g., emotional tone, hyperbole). High-risk topics frequently associated with misinformation. Identification of repeat sources spreading disinformation.Trends in user engagement with fake vs. real news. Analytics dashboard showing fake news trends over time, topic, or region.

**5. Flowchart of Project Workflow**

A diagram of a data collection

AI-generated content may be incorrect.

# 6. Dataset Description

# 1. Name: Fake and Real News Dataset

# 2. Source: Kaggle – Clément Bisaillon

# 3. Type of Data:

# Textual Data: News headlines and full articles

# Categorical Labels: “FAKE” and “REAL”

# Metadata: Publication date and subject (e.g., politics, world)

Sample df.head() Output

import pandas as pd

# Simulated df.head() from the dataset

data = {

'title': [

"Donald Trump Sends Out Embarrassing New Year’s Eve Message",

"Drunk Bragging Trump Staffer Started Russian Collusion Investigation",

"Sheriff David Clarke Becomes An Internet Joke",

"Trump Is Trying To Avoid Paying Taxes Again",

"Trump Just Lost His Case On Illegal Voting"

],

'text': [

"Donald Trump just couldn’t wish all Americans a Happy New Year...",

"An inebriated Trump staffer made a careless admission...",

"Sheriff Clarke’s attempt at being a tough guy backfired hilariously...",

"President Trump is exploring another tax loophole...",

"A federal judge ruled against Trump’s baseless claims..."

],

'subject': ["politicsNews", "politicsNews", "politicsNews", "politicsNews", "politicsNews"],

'date': ["December 31, 2017", "December 30, 2017", "December 29, 2017", "December 28, 2017", "December 27, 2017"],

'label': [1, 1, 1, 1, 1] # 1 = FAKE

}

df = pd.DataFrame(data)

print(df.head())

Structure:

title: Headline of the news article (string)

text: Full article content (string)

subject: News topic/category (e.g., politics, world)

date: Date of publication (string)

label: Ground truth (1 = FAKE, 0 = REAL)

# 7. Data Preprocessing

**1. Handle Missing Values**

Check for missing values in columns like title, text, subject, and date.

Options:

Drop rows with missing critical data (especially in text).

Or fill missing subject or date with placeholders like "unknown" or median/most frequent values.

Example:

df.dropna(subset=['text'], inplace=True)

df['subject'].fillna('unknown', inplace=True)

**2. Remove Duplicates**

Duplicate articles can bias the model.

Drop duplicates based on title and text.

Example:

df.drop\_duplicates(subset=['title', 'text'], inplace=True)

**3. Handle Outliers**

For text data, outliers might be unusually long or short articles.

Optionally, filter out articles with extremely low or high word counts.

Example:

df['text\_length'] = df['text'].apply(lambda x: len(x.split()))

df = df[(df['text\_length'] > 20) & (df['text\_length'] < 2000)]

**4. Feature Encoding**

Encode categorical variables if used in the model:

Example: subject can be label encoded or one-hot encoded.

For NLP, text data will be converted using embeddings or vectorizers (TF-IDF, Word2Vec, BERT).

Example:

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

df['subject\_encoded'] = le.fit\_transform(df['subject'])

**5. Text Preprocessing (important for NLP)**

Lowercae conversion

Remove punctuation, special characters

Remove stopwords

Tokenization and lemmatization/stemming

Example with SpaCy:

import spacy

nlp = spacy.load('en\_core\_web\_sm')

def preprocess\_text(text):

doc = nlp(text.lower())

tokens = [token.lemma\_ for token in doc if not token.is\_stop and token.is\_alpha]

return ' '.join(tokens)

df['clean\_text'] = df['text'].apply(preprocess\_text)

**6. Scaling**

Typically, scaling applies to numerical features.

For text features converted to vectors (TF-IDF), scaling is not mandatory.

If combining with numeric features (like article length), use:

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

df['text\_length\_scaled'] = scaler.fit\_transform(df[['text\_length']])

# 8. Exploratory Data Analysis (EDA)

**1. Class Distribution**

Visualize counts of FAKE vs. REAL news articles.

Identify if dataset is balanced or imbalanced.

Use bar plots or pie charts.

**2. Text Length Analysis**

Calculate word count or character count per article.

Plot distributions for FAKE and REAL separately.

Look for patterns like fake news being shorter or longer.

**3. Topic/Subject Analysis**

Count articles by subject/category (politicsNews, worldnews, etc.).

Compare subject frequency across FAKE and REAL classes.

**4. Publication Date Trends**

Plot number of articles over time.

Check for spikes in fake news publication during events or periods.

**5. Correlation Analysis**

Correlate numerical features (e.g., text length) with labels.

Use correlation coefficients (Pearson, Spearman).

Identify if certain features strongly relate to fake/real labels.

**6. Word Frequency & Common Terms**

Generate word clouds or frequency plots for each class.

Identify words highly associated with fake or real news.

**7. Sentiment Analysis**

Analyze sentiment scores distribution in FAKE vs. REAL.

Detect if fake news tends to have more extreme sentiments.

**8. N-gram Analysis**

Extract common bi-grams or tri-grams in fake and real news.

Look for patterns in phrasing or repeated expressions.

**1. Univariate Analysis (Single variable)**

Purpose: Understand the distribution and nature of each feature individually.

**a) Target Variable (label)**

Plot: Bar chart of FAKE (1) vs REAL (0)

Insight: Check for class imbalance.

**b) Text Length**

Feature: Word or character count of text

Plot: Histogram or KDE plot

Insight: Fake news often has shorter articles than real news.

**c) Subject**

Plot: Count plot (bar chart)

Insight: Which topics are most common? Are some topics more prone to fake news?

**2. Bivariate Analysis (Two variables)**

Purpose: Explore the relationship between two variables, often between features and the label.

**a) Text Length vs Label**

Plot: Boxplot or Violin plot

Insight: Real articles may be more detailed; visualize average lengths by class.

**b) Subject vs Label**

Plot: Stacked bar chart

Insight: Some subjects (e.g., politics) may have a higher ratio of fake news.

**c) Sentiment Score vs Label (Optional with sentiment analysis)**

Plot: Strip plot or swarm plot

Insight: Fake news may show more extreme sentiment.

**3. Multivariate Analysis (Three or more variables)**

Purpose: Understand interactions between multiple features.

**a) Word Count + Subject + Label**

Plot: Faceted boxplots or heatmaps

Insight: Explore whether certain topics have consistently longer/shorter fake or real articles.

**b) Top Keywords per Label per Subject**

Approach: Group data by subject and label, extract top N words per group

Insight: Reveal topic-specific fake news language patterns.

**c) Correlation Matrix (Numeric Features)**

Plot: Heatmap

Insight: Correlate text length, sentiment, TF-IDF scores with labels.

**Optional Visual Tools:**

Word clouds (separate for FAKE and REAL)

TF-IDF feature importance for classification

N-gram frequency comparison

**Data Visualization**

Matplotlib:

Basic plots like bar charts, histograms, pie charts

Useful for text length, label distribution

Seaborn:

Advanced statistical plots: boxplots, heatmaps, violin plots, strip plots

Great for visualizing relationships and distributions

Plotly (Optional):

Interactive plots for exploratory dashboards

# 9. Feature Engineering

**1. Feature Creation**

Extract new features from the raw text to enhance model learning:

Text-based features:

text\_length: Total number of characters or words.

avg\_word\_length: Average length of words.

num\_sentences: Number of sentences.

num\_punctuations: Count of punctuation marks.

num\_uppercase\_words: Count of fully capitalized words (e.g., SHOCKING).

num\_stopwords: Count of stopwords in the article.

Lexical/Semantic features:

sentiment\_score: Sentiment polarity using TextBlob or VADER.

subjectivity\_score: Degree of opinionated language.

named\_entity\_count: Number of named entities using SpaCy.

**2. Feature Selection**

Pick the most informative features and discard irrelevant or redundant ones

Statistical methods:

Chi-square test: To assess the relationship between features and the label.

ANOVA/F-test: To evaluate numerical features' variance with class labels.

Model-based selection:

Feature importance from models: Use Random Forest, XGBoost, or Logistic Regression to rank features.

Recursive Feature Elimination (RFE): Iteratively remove least important features.

Dimensionality reduction (for vectorized text):

TF-IDF + TruncatedSVD or PCA: Reduce high-dimensional TF-IDF vectors while preserving most variance.

**3. Transformation Techniques**

Convert and scale features for better performance and convergence:

Text Vectorization:

TF-IDF (Term Frequency-Inverse Document Frequency): Most commonly used for NLP tasks.

Count Vectorizer: Simpler form, useful for small models.

Word embeddings (optional): GloVe, Word2Vec for deep learning models.

Encoding categorical features:

Label Encoding: For simple categorical fields like subject.

One-hot Encoding: When model is sensitive to categorical scale.

Scaling numerical features

StandardScaler / MinMaxScaler: Normalize features like text length or sentiment scores for ML algorithms like SVM or Logistic Regression.

**Feature Scaling**

A. Select Numeric Features to Scale

numerical\_features = df[['text\_length', 'avg\_word\_length', 'num\_sentences', 'sentiment\_score', 'subjectivity\_score', 'named\_entity\_count']]

B. Apply Scaling

StandardScaler (Z-score Normalization):

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaled\_numerical = scaler.fit\_transform(numerical\_features)

Or use MinMaxScaler if required:

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

scaled\_numerical = scaler.fit\_transform(numerical\_features)

# 10. Model Building

**1. Train-Test Split**

Goal: To divide the dataset into training and testing sets so that we can evaluate model performance.

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(final\_features, y, test\_size=0.2, random\_state=42)

final\_features: Combined TF-IDF and scaled numeric features

y: Target label (FAKE=1, REAL=0)

test\_size=0.2: 20% data reserved for testing

random\_state: Ensures reproducibility

**2. Baseline Model: Logistic Regression**

Why? It’s simple, fast, and interpretable. Used as a benchmark.

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import classification\_report

model = LogisticRegression()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

print(classification\_report(y\_test, y\_pred))

Explanation: This fits a linear boundary to classify news as fake or real.

Output: Accuracy, Precision, Recall, F1-score

**3. Advanced Machine Learning Models**

**A. Random Forest Classifier**

Why? It handles non-linear patterns and gives feature importance.

from sklearn.ensemble import RandomForestClassifier

rf\_model = RandomForestClassifier()

rf\_model.fit(X\_train, y\_train)

rf\_pred = rf\_model.predict(X\_test)

print(classification\_report(y\_test, rf\_pred))

Explanation: Ensemble of decision trees, good for complex patterns.

**B. XGBoost**

Why? It's a powerful boosting algorithm with high accuracy.

from xgboost import XGBClassifier

xgb\_model = XGBClassifier(use\_label\_encoder=False, eval\_metric='logloss')

xgb\_model.fit(X\_train, y\_train)

xgb\_pred = xgb\_model.predict(X\_test)

print(classification\_report(y\_test, xgb\_pred))

Explanation: Boosts weak learners to form a strong model, handles imbalance better.

**C. Support Vector Machine (SVM)**

Why? Effective in high-dimensional spaces (like TF-IDF features).

from sklearn.svm import LinearSVC

svm\_model = LinearSVC()

svm\_model.fit(X\_train, y\_train)

svm\_pred = svm\_model.predict(X\_test)

print(classification\_report(y\_test, svm\_pred))

Explanation: Finds the optimal separating hyperplane between fake and real news.

**4. Deep Learning Model (LSTM)**

Why? Learns long-term patterns in sequential data (text).

from keras.models import Sequential

from keras.layers import Embedding, LSTM, Dense

from keras.preprocessing.sequence import pad\_sequences

from keras.preprocessing.text import Tokenizer

# Tokenization

tokenizer = Tokenizer(num\_words=5000)

tokenizer.fit\_on\_texts(df['text'])

X\_seq = tokenizer.texts\_to\_sequences(df['text'])

X\_padded = pad\_sequences(X\_seq, maxlen=300)

# Split

X\_train\_dl, X\_test\_dl, y\_train\_dl, y\_test\_dl = train\_test\_split(X\_padded, y, test\_size=0.2)

# Model

model = Sequential()

model.add(Embedding(input\_dim=5000, output\_dim=64, input\_length=300))

model.add(LSTM(64))

model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

model.fit(X\_train\_dl, y\_train\_dl, epochs=3, batch\_size=64)

Explanation: LSTM understands sequence and context better than traditional models.

**5. Transformer-Based Model (BERT)**

Why? Pretrained deep model that understands language contextually. Best for NLP.

from transformers import BertTokenizer, TFBertForSequenceClassification

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.losses import BinaryCrossentropy

tokenizer = BertTokenizer.from\_pretrained('bert-base-uncased')

# Tokenization

train\_encodings = tokenizer(list(X\_train\_text), truncation=True, padding=True)

test\_encodings = tokenizer(list(X\_test\_text), truncation=True, padding=True)

# Load model

bert\_model = TFBertForSequenceClassification.from\_pretrained('bert-base-uncased')

# Compile

bert\_model.compile(optimizer=Adam(learning\_rate=2e-5),

loss=BinaryCrossentropy(from\_logits=True),

metrics=['accuracy'])

# Fit

bert\_model.fit(train\_encodings, y\_train, epochs=3, batch\_size=16)

Explanation: BERT is context-aware and pre-trained on massive corpora. Fine-tuning it provides state-of-the-art performance.

# 11. Model Evaluation

**1. Evaluation Metrics**

Accuracy:

Measures overall correctness:

Accuracy = (TP + TN) / (TP + TN + FP + FN)

Precision:

How many predicted fakes were actually fake?

Precision = TP / (TP + FP)

Recall (Sensitivity):

How many actual fakes were correctly identified?

Recall = TP / (TP + FN)

F1-Score:

Harmonic mean of precision and recall. Useful for imbalanced data.

F1 = 2 \* (Precision \* Recall) / (Precision + Recall)

Confusion Matrix:

Shows TP, TN, FP, FN in a 2x2 matrix for better understanding of model performance.

**2. Evaluation Code Example**

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

y\_pred = model.predict(X\_test)

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print("Classification Report:\n", classification\_report(y\_test, y\_pred))

print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred))

**3. Visual Evaluation (Optional)**

ROC Curve: Plots true positive rate vs. false positive rate

AUC Score: Area under ROC curve. Closer to 1 is better.

from sklearn.metrics import roc\_auc\_score, roc\_curve

import matplotlib.pyplot as plt

fpr, tpr, \_ = roc\_curve(y\_test, model.predict\_proba(X\_test)[:,1])

plt.plot(fpr, tpr)

plt.title("ROC Curve")

plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

plt.show()

print("AUC Score:", roc\_auc\_score(y\_test, model.predict\_proba(X\_test)[:,1]))

**4. Interpretation**

High accuracy with high precision & recall → model is performing well.

Low precision → too many false positives.

Low recall → fake news is being missed (false negatives).

Balanced F1-score → good for both precision & recall when dataset is imbalanced.

# 12. Deployment

**1. Streamlit Cloud**

Use case: Quick and interactive web apps for ML projects.

Steps:

Save your model (.pkl or .h5).

Create app.py with Streamlit UI.

Add requirements.txt for dependencies.

Push to GitHub and connect to streamlit.io.

Pros: Simple UI, live text prediction, ideal for demos.

import streamlit as st

st.title("Fake News Detector")

text = st.text\_area("Enter news text")

if st.button("Predict"):

prediction = model.predict([text])

st.write("Prediction:", "FAKE" if prediction[0] else "REAL")

**2. Gradio**

Use case: Instant browser-based demo.

Steps:

Install with pip install gradio.

Create interface with gr.Interface().

Launch locally or share online.

import gradio as gr

def predict(text):

return "FAKE" if model.predict([text])[0] else "REAL"

gr.Interface(fn=predict, inputs="text", outputs="text", title="Fake News Detector").launch()

Pros: One line hosting, supports Hugging Face integration.

**3. Hugging Face Spaces**

Use case: Community sharing of ML apps.

Supports: Gradio and Streamlit

Steps:

Fork Gradio/Streamlit template.

Upload your code, model, and requirements.txt.

Hugging Face hosts it for free.

Pros: Free, visible to the NLP community.

**4. Flash (by Lightning AI)**

Use case: Production-grade AI app deployment.

Steps:

Build pipeline with PyTorch Lightning.

Use Flash components for preprocessing + inference.

Pros: Fast GPU inference, scalable apps.

**5. Render**

Use case: Full backend deployment (API + frontend).

Steps:

Create Flask/FastAPI app.

Deploy via GitHub on Render.

Add model file + dependencies.

Pros: Suitable for API integration.

**6. Deta Space**

Use case: Lightweight apps or APIs for free.

Steps:

Create FastAPI or Python script.

Use deta deploy to host.

Pros: Great for lightweight, fast serverless APIs.

**13. Source code**

**Code-1:**

import pandas as pd

import numpy as np

import re

import nltk

from nltk.corpus import stopwords

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import MultinomialNB

from sklearn.metrics import accuracy\_score, classification\_report

# Load Dataset

df = pd.read\_csv("fake\_news\_dataset.csv") # Adjust file path as needed

# Preprocessing: Clean text data

nltk.download('stopwords')

stop\_words = set(stopwords.words('english'))

def clean\_text(text):

text = str(text).lower() # Lowercase

text = re.sub(r'\W', ' ', text) # Remove special characters

text = re.sub(r'\s+', ' ', text) # Remove extra spaces

text = ' '.join([word for word in text.split() if word not in stop\_words]) # Remove stopwords

return text

df["clean\_text"] = df["text"].apply(clean\_text) # Assuming "text" is the column with news articles

# Vectorization: Convert text into numerical format

vectorizer = TfidfVectorizer(max\_features=5000) # TF-IDF approach

X = vectorizer.fit\_transform(df["clean\_text"]).toarray()

y = df["label"] # Assuming "label" column contains Fake (0) vs Real (1) classification

# Train-Test Split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Model Training: Using Naive Bayes classifier

model = MultinomialNB()

model.fit(X\_train, y\_train)

# Prediction

y\_pred = model.predict(X\_test)

# Evaluation

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy: {accuracy:.2f}")

print("Classification Report:\n", classification\_report(y\_test, y\_pred))

# Predicting on a new sample

new\_text = ["Breaking news! Scientists discover AI with human-level sarcasm!"]

new\_text\_clean = [clean\_text(text) for text in new\_text]

new\_text\_vectorized = vectorizer.transform(new\_text\_clean).toarray()

prediction = model.predict(new\_text\_vectorized)

print("Prediction (0=Fake, 1=Real):", prediction)

**Code-2:**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load your dataset

df = pd.read\_csv("fake\_news\_dataset.csv")

# Example 1: Bar plot of news source distribution

plt.figure(figsize=(10, 5))

sns.countplot(y=df["source"], order=df["source"].value\_counts().index)

plt.title("Distribution of News Sources")

plt.xlabel("Count")

plt.ylabel("Source")

plt.show()

# Example 2: Word count distribution in article text

df["word\_count"] = df["text"].apply(lambda x: len(str(x).split()))

plt.figure(figsize=(8, 5))

sns.histplot(df["word\_count"], bins=30, kde=True)

plt.title("Distribution of Word Count in Articles")

plt.xlabel("Word Count")

plt.ylabel("Frequency")

plt.show()

# Example 3: Fake vs Real news distribution

plt.figure(figsize=(6, 4))

sns.countplot(x=df["label"]) # Assuming 'label' column marks fake (0) vs real (1) news

plt.title("Distribution of Fake vs Real News")

plt.xlabel("News Type (Fake=0, Real=1)")

plt.ylabel("Count")

plt.show()

# 14. Future scope

**1. Enhanced Real-Time Detection Systems**

Future Scope: Integration into news platforms and social media to detect and flag fake news instantly.

Impact: Limits the spread of misinformation before it gains traction.

**2. Multilingual and Cross-Cultural Fake News Detection**

Future Scope: Development of NLP models that work across languages, dialects, and cultural contexts.

Impact: Global reach, especially valuable in multilingual nations or during global events like pandemics or elections.

**3. Deepfake and Multimodal Misinformation Detection**

Future Scope: Combining NLP with computer vision and audio analysis to detect fake news that includes text, images, and videos.

Impact: Comprehensive verification across multiple media formats.

**4. Integration with Blockchain for Provenance Tracking**

Future Scope: Use blockchain to verify the source and history of news articles and media.

Impact: Transparent and tamper-proof content verification systems.

**5. Personalized Misinformation Risk Alerts**

Future Scope: AI models that assess an individual's exposure to misinformation and provide tailored alerts or fact-checks.

Impact: Increases individual awareness and digital literacy.

**6. Policy-Making and Legal Framework Support**

Future Scope: Assisting governments and organizations in shaping regulations with data-driven insights on fake news trends.

Impact: Smarter governance and better-prepared legal systems to handle misinformation.

**7. Educational Tools and Digital Literacy Platforms**

Future Scope: Developing educational platforms using NLP to teach critical thinking and fake news identification.

Impact: Long-term societal resilience against misinformation.

**8.Collaboration with Journalistic and Fact-Checking Bodies**

Future Scope: Tools that assist journalists in validating sources and content before publication.

Impact: Strengthens the integrity of journalism.

# 13. Team Members and Roles

**M.Sanjai pravin** worked in

Data cleaning

Preprocessing

Feature encoding

**V.Sanjay kumar** worked in

EDA

Statistical insights

Visualization

**K.Sathya Sri worked in**

Feature engineering

Model development

**S.Sathya Priya** worked in

Evaluation

Reporting